SYSTEM AND METHOD FOR REAL-TIME GENERATION OF SOFTWARE TRANSLATION

FIELD OF THE INVENTION

[0001] The subject matter of this application is related to the subject matter of U.S. Provisional Serial No. 60/498,282, filed August 28, 2003, from which application this application claims priority.

FIELD OF THE INVENTION

[0002] The invention relates to the field of computer software, and more particularly to techniques for automatically generating language-specific versions of application or other software.

BACKGROUND OF THE INVENTION

[0003] The software industry is a global business whose markets encompass various countries, many of which have different languages. For software vendors who intend to ship product into foreign countries, the process of localizing software into different languages can often only begin after the core software development cycle has ended. Vendors have often been unable to translate and then test translations before code is released to those foreign clients. Translated applications may be delivered for example to non-U.S. clients several months later than original US code.

[0004] In the industry, the standard approach to delivering a translated localized product is consequently to translate and package each localized product individually. The release of localized products may further be prioritized by importance of the target market. Thus, there may be individually packaged products of productivity software for Germany, Thailand and other countries or markets. The German market may have a higher priority than the Thai or other market. Hence, the German product may be released simultaneously or shortly after the U.S. product, while the Thai or other product may be released months later. This is called a tiered approach.

[0005] There are several downsides to the tiered approach. Individually packaged products may often have different defects. Having a set of differing versions of code may make it more difficult to troubleshoot and resolve issues. If an issue is identified and resolved, it may only be resolved for one individual language package. The tiered approach may also require the localization team to spend months or longer with one product at a time, rather than delivering all localized products simultaneously. Development resources may therefore be taken away from subsequent releases.

[0006] Another, less common approach to delivering localized software products is to release one core source code version to all customers, and add components that display translations at run time if the source code detects a regional setting that is different from the regional setting of the source code. The advantages to this approach include that only one version of source code needs to be supported and maintained. The translated components may consist largely or only of resource-only dynamic link libraries (dlls) that contain no additional functionality.

[0007] Therefore, those components do not require processing time that could have an impact on application performance. Typically, these components do not cause application bugs or issues similar to issues caused by source code. However, even vendors employing this run-time approach usually focus on releasing one localized version after another on a per-product basis, and on a defined schedule. No more than one localized product line may be shipped simultaneously.

[0008] A need consequently exists for a translation process that allows for a real time translation release of all available products, without a need to prioritize languages or markets or to schedule staggered releases of the software product.

SUMMARY OF THE INVENTION

[0009] The invention overcoming these and other problems in the art relates in one regard to a system and method for real-time translation of applications and other software products, in which a parsing engine may accept a finished application or other source code as an input in an original language, and automatically generate one or more language-translated versions of the software product. According to the invention in one regard, the language-translated version of the software may have internationalization rules automatically enforced, to help prevent errors and ensure product consistency across markets. Because the parsing engine and related components may cooperate to quickly and consistently transform source code to various languages for various destination markets, translated software packages may be generated and deployed in real time with the original code release. Vendors may thus release software products without having to allow for additional localization time in the coding cycle, and without using a tiered approach to different markets.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The invention will be described with reference to the accompanying drawings, in which like numbers reference like elements.

[0011] Fig. 1 illustrates an overall network architecture in which an embodiment of the invention may operate.

[0012] Fig. 2 illustrates the generation of a translated output based on a parsing action according to an embodiment of the invention.

[0013] Fig. 3 illustrates internationalization processing according to an embodiment of the invention.

[0014] Fig. 4 illustrates a flowchart of overall translation processing according to an embodiment of the invention.

DETAILED DESCRIPTION OF EMBODIMENTS

[0015] An illustrative environment in which an embodiment of the invention may operate is shown in Fig. 1, in which a software developer, programmer or others may operate a developer workstation 108 communicating with a code database 104 and other resources to translate textual strings or other language-specific portions of a source component 102 into one or more target languages. The source component 102 may be or include, for instance, a software application such as a word processing, spreadsheet, database or other application, a portion or component of a software application such as a function, module, class or routine, or other code or software resources.

[0016] According to the localization processing of the invention in one regard, a developer or other user may transmit the subject source component 102 to a parsing engine 110. The parsing engine 110 may in turn communicate with the code database 104 hosting one or more translation resources used to generate language-tailored versions of the source component 102. According to embodiments of the invention, the code database 104 may be established for the storage, translation and dissemination of original application and other code and other resource. As illustrated, the code database 104 may among other things host an existing translation list 106 of all text strings 112, such as English language or other strings, derived from one or more source component 102 that have been previously translated for code localization or other purposes.

[0017] The code database 104 may allow for flexible activation of products and languages according to countries or markets, since not all source code products may be suitable for all global markets. The existing translation list 106 may therefore include the ability to adjust the entries depending on source and target languages and intended markets. The existing translation list 106 may, for example, contain, alter or expose different sets of source text depending on the source or target country or language.

[0018] According to embodiments of the invention, if a developer or other user checks in a new or modified source component 102 for processing, parsing engine 110 may be activated and scan the source component 102 for textual expressions or other language-dependent fields which may be presented to the application user. For example, the parsing engine 110 may detect modules or routines such as those which may be labeled “GetMessages”, “BuildMessages” or “FormCaptions”, as well as other resources, files or content. Other modules, routines, resources or whole or partial code may be scanned or detected.

[0019] Once the parsing engine 110 has identified all text strings 112 embedded in candidate modules or routines, the parsing engine 110 may query the existing translation list 106 to determine whether any of the text strings 112 have been translated before. In embodiments, one or more of the text strings 112 may be have been translated or stored in connection with the subject source component 102, or with other source components but made available for reuse.

[0020] If all text strings 112 in the current source component 102 have been translated before, no further action may be taken. If some of the text strings 112 have not been translated, those strings may be placed in a separate untranslated text string table 114. In embodiments developers or others may have the ability to access the untranslated text string table 114, for instance via a front-end interface, and retrieve any untranslated text strings 112, for instance in .txt or other format.

[0021] Once retrieved, the developer or others may send those text strings 112 to an appropriate translation facility for translation. That facility may be or include a translation agency or bureau performing manual, automated or other translation service. After translation, translated text strings 116 may be returned and communicated for storage in code database 104, for instance in a TRADOS translation format known to persons skilled in the art, or in other memory, media or format. TRADOS "memory" for example stores every pair of translated source and target text strings for consistent use, ensuring that identical text strings 112 need only be translated once, regardless of their origin. The developer or other user in embodiments may select which target language in which the translated text strings 116 may be retrieved, via parsing engine 110 and a front-end interface executing on developer workstation 108, or otherwise.

[0022] If a decision to translate a new source component 102 is made, the source component 102 or related components in a complete application or other product may be added to the code database 104. The code database 104 may check for any new untranslated text strings 112 within the new source component 102 immediately, and generate one or more translated text strings 116 in very little time if such strings are detected. Likewise, in embodiments if the translation of certain software components or products is discontinued, the corresponding translated text strings 116 may be promptly deactivated in the database. Once any new translated text strings 116 have been retrieved, they may be added to the existing translation list 106 and will not require translation again.

[0023] According to the invention in another regard, once translated text strings 116 have been generated and projects or stages of projects have been built successfully, the parsing engine 110 communicating with code database 104 may generate run-time translation resources 120 for the subject source component 102. In embodiments, the run-time translation resources 120 may be, include or interface to resource-only dlls from all related software components or modules, such as Visual Basic, C++, Java or other code, modules or other resources. The run-time translation resources 120 may be or include the data or other components that will contain and display translated text at runtime. In embodiments, the run-time translation resources 120 may not contain any additional functionality. This in one regard may help to ensure that the end-user's computer is not unnecessarily loaded. Furthermore, in another regard there may be little or no functionality in the run-time translation resources 120 that can interfere with or cause errors in the overall software product's functionality.

[0024] In embodiments, the parsing engine 110 may generate at least two types of run-time translation resources 120 for each piece of translated software. Those types of run-time translation resources 120 may be or include at least dlls for pseudo translation, and translation dlls (sometimes referred to as enu.dlls). Dlls for pseudo translation may place a placeholder expression, such as asterisks or other expressions, in front of strings displayed at runtime if the user's regional settings are changed to an unsupported language. Such flags may help developers review and test the translated product further.

[0025] Translation dlls may likewise be generated and provided to the developer or other users. Those translation dlls may employ a naming convention, for example such that every dll,

.exe, .ocx or other component of the source code receives a corresponding component with the same name plus a fixed extension, such as enu.dll. Thus, for example, for "Powerchart.exe" the corresponding translation component may be designated Powerchart_enu.dll. The translation dlls (enu.dlls) may then be loaded into a localization tool 122, such as the commercially available Visual Localize (VisLoc), or other off-the-shelf or other localization tool. The localization tool 122 may retain previous versions of the same source component 102 to assure that strings that have been previously translated do not need to be translated again.

[0026] Once the new components have been uploaded, the localization tool 122 may allow the export of all text strings that have not been translated since the upload of the last version. This string export can be translated automatically, for example by using the commercially available TRADOS translation memory that retains all the strings that are still untranslated in coherent storage (e.g. VisLoc), as noted above. The translations may be imported into the localization tool 122 and translation dlls may be generated. The translation dlls may be named so that the appropriate regional settings can detect and call them. For example for German-language products, the translation dlls may be renamed to "_deu.dll". Once generated and named, the run-time translation resources 120 may be communicated to the software factory or other facility with validated translated components for packaging. Once translated, corrected and transmitted, the product or products incorporating translated text strings 116 may be assembled and packaged, for instance automatically according to a bill of material.

[0027] As likewise illustrated in Fig. 1, when an end user in a set of end users 130 downloads a new release of a product generated according to the invention, in embodiments they may be first prompted to download the original, non-localized application 128 in an original language version. After a successful download via network 126, which may be or include the

Internet or related connections, the end user in the set of end users 130 may be prompted to download appropriate translation material for their market to import or install to generate the ultimate localized application 124.

[0028] In configuring that download, the vendor of the product may be able to determine which language version to provide to the end user, based on login or other information supplied by or related to the end user. For example, in embodiments the regional settings on the end user's computer may be detected to identify a target language or languages. In other embodiments, the user may be queried via a dialog box or otherwise for their language preference. In the installation process, a link to a site in or connected to network 126 for downloading run-time translation resources 120 may appear only after the successful download or registration of the original non-localized application 128, itself. According to the invention in one regard, the end user therefore only download run-time translation resources 120 after successfully downloading the original-language non-localized application 128, as opposed to accidentally or intentionally only downloading run-time translation resources 120 by themselves, with no executable functionality.

[0029] According to the invention in another regard, to permit the real-time release of localized software without interfering with the engineering of underlying code, the translation and validation processes of the source component 102 may thus be effectively separated. Validation of translated applications or components therefore may be more efficient when source code is mature, and it becomes highly likely or certain that new functionality will actually be incorporated in the source code of non-localized application 128. This may typically be the case a few weeks before the release date of source software, at which point in the development cycle it may not be feasible to conventionally translate all applications and test the various translations of those applications for prompt release.

[0030] As illustrated in Fig. 2, once a source component destined for incorporation into an ultimate localized application 124 has been translated and debugged, the run-time translation resources 120 may present or contain a language-tailored version of user prompts, dialog boxes and other user interface or other text. So for instance a user query originally appearing in English in the source component 102 may be transparently converted to equivalent translated expressions in German or other languages communicated, referenced or embedded in run-time translation resources 120, as shown.

[0031] According to the invention in another aspect, code internationalization standards may in embodiments be enforced or followed to assure that localized products are validated to comparable quality as the original source component 102. If quality checks are not enforced, text may unintentionally appear in untranslated form in the wrong language to the end user, or certain functionality may not work as intended.

[0032] As illustrated in Fig. 3, the parsing engine 110 according to the invention may in embodiments therefore additionally analyze the syntax of a given source component 102. The parsing engine 110 may for those purposes in embodiments also include both a parsing log 118 and a syntax module 132. Parsing log 118 may reflect the textual strings 112 detected in source component 102, the target language or languages to be generated and other information. The syntax module 132 may contain or be capable of accessing code internationalization standards or rules for various programming languages, such as Visual Basic, Visual C++, Java or other languages or platforms. Those standards or rules may include, for example, International

Standards Organization (ISO), European Union (EU) or other standards, requirements or protocols.

[0033] The syntax module 132 may scan the translation or translation-in-progress of source component 102 and detect translation irregularities. When irregularities are trapped in this fashion, the developer or others who checked the source component 102 may receive a notification flag 134 and a copy of the parsing log 118. That developer then may review and correct the issue, and check the updated source component 102 in again. If no further syntactical issues are found, the translated software can be output successfully. If internationalization errors are returned or returned again for a source component 102 which was checked in, the code build may not complete successfully. In embodiments, parsing engine 110 and other components according to the invention may also be configured to except entire products or single components of projects from being parsed and checked for internationalization compliance, if circumstances dictate.

[0034] According to the invention in one other regard, code internationalization and the creation of functionality may therefore be integrated. The automated quality control process according to embodiments of the invention may fail code that is not properly internationalized, allowing developers and vendors to trap and avoid internationalization issues from the beginning. Additional developer teams dedicated to code internationalization may therefore be unnecessary.

[0035] Overall translation processing according to an embodiment of the invention in one regard is illustrated in Fig. 4. In step 402, translation processing may begin. In step 404, the original source component 102 may be developed, for instance by a developer or developer team. In step 406, a host name (hnam) may be identified for the code database 104 into which source component 102, translation text strings and other code or resources may be stored, for instance in a relational or other database. In step 408, the source component 102 may be checked in to the code database 104, for instance by a developer via a secure or other connection over the Internet, or otherwise.

[0036] In step 410, the source component 102 may be scanned for the presence of existing translated text strings 116, for instance by checking existing translation list 106. If translated text strings 116 corresponding to text in the source component 102 are available, processing may proceed to step 424 in which run-time translation resources 120, such as dll files or others, may be imported. After those resources are imported, processing may proceed to step 418 in which run-time translation resources 120 may be received. In step 420, the localized component 124 may be built or rebuilt to include the language-specific run-time translation resources appropriate to the given market or country.

[0037] If in step 410 no existing translated text strings 116 are detected, processing may proceed to step 412, in which parsing engine 110 may process text strings 112 to generate translated text strings 116. If parsing engine 110 fails to associate appropriate translated text strings 116 to text strings 112 or otherwise fails to complete that operation in step 414, processing may return to step 408 to receive a check-in of an updated or other source component 102. In embodiments developers or others may be notified of that failure. If the parsing engine however successfully parses text strings 112 in step 412, processing may proceed to step 416 in which one or more parts of localized component 124 may be built, followed by receipt of any remaining run-time translation resources 120 and rebuilding of localized component 124 in step 420.

[0038] In step 424, after the run-time translation resources 120 are imported, processing may likewise proceed to step 426 in which additional run-time translation resources, such as graphically enabled translated text strings 116, may be generated. For instance, translated text strings 116 may be combined with bit-mapped or other graphics to display dialog boxes, warning messages or other content. In step 428, the source component may be further developed or debugged using localization tool 122, for instance the commercially available VisLoc or other tools or utilities. In step 430, the build generated by localization tool 122 may be tested for compliance with internationalization standards, for instance internal or external internationalization protocols or standards.

[0039] In step 432, corrections or updates to the localized component 124 may be performed as appropriate. In step 434, testing for correct internationalization may be repeated, followed by additional corrections to translated text strings 116 or other components. In step 436 any remaining corrections may be made. In step 438 the localized component 124 may be reloaded and subjected in step 440 to a regression test. In step 442 assuming no further internationalization issues remain the localized component 124 may be promoted to finished status. In step 454 localized component 124 may be delivered to or incorporated in a vendor factory package and shipped, made available for download or otherwise distributed.

[0040] In step 444, help files associated with source component 102 may be retrieved, followed by translation of those files, for instance by using a translation bureau or other facility, in step 446. In step 448, the localized component 124 may be compiled with the translated text strings 116 related to those help or other files. In step 450, operation of the localized component 124, such as an application program or other language-tailored product, may be test for accurate presentation of help or other files. In step 452, the localized component may be recompiled to update any corrections or modifications, and in step 454 the vendor factory package of localized component 124 may be completed, shipped, made available for download or otherwise distributed.

[0041] In step 456, database scripts or related resources associated with localized component 124 may be accessed and retrieved in step 458 for internationalized translation or other processing. Those scripts, files or other resources may include, for instance, character separated interface (CSV) databases or supporting scripts. In step 460, concatenation of the localized component 124 with supporting CSV databases or related Perl or other scripts may be performed. In step 462, translation captions may be copied to a ".doc" or other file. In step 464, a translation of text strings 112 contained in or related to CSV database entries or related scripts may be performed to generate translated text strings 116 or other output. In step 466, the translated text strings 116 or other translation output may be overlayed in spreadsheet (for example Microsoft Excel™) or other format.

[0042] In step 482 CSV or other database files may be built, and in step 484 those files may be communicated to a backend storage server or other resource, for instance via file transfer protocol (FTP) over the Internet or other networks. In step 486, upload scripts may be executed. In step 488, the CSV or other database, database entries, supporting scripts or other resources or modules may be tested, and after validation in step 490 those resources may be attached to the localized software product package or other object. In step 492, after testing or validation is complete, the assembled software package incorporating localized component 124 may be promoted to final or shipping version status, after which that package may be packaged for shipment, made available for download or otherwise distributed in step 494. In step 496, an install test may be performed to ensure that public customers may connect to code database 104 or other sites and retrieve and install software packages incorporating appropriate localized component 124.

[0043] In step 468, according to the invention in another regard an application task request (ATR), which may be a code identifying software applications or other software, along with codesets which according to the invention may be or include entries on a database table that may be called by applications, may be accessed. In step 470, data from application specifications or other codesets may be pulled from a database. In step 472, an access table may be built on the code database 104 or other database. In step 474, the resulting textual description or display information may be copied to a ".doc" or other file or entry. In step 476, the codesets or other data may be converted to textual format. In step 478, that set or sets of text strings 112 may be translated into a target language. In step 480, translated text strings 116 or other translated display data corresponding to that original data may be added to the code database 104 or other storage. Processing may then proceed to steps 482 through 496, operating as described above.

[0044] The foregoing description of the invention is illustrative, and modifications in configuration and implementation will occur to persons skilled in the art. For instance, while the invention has generally been described in terms of a single code database 104 servicing developer language conversions and end-user localization, in embodiments multiple databases or other data stores could be employed. In embodiments, for example one or more database may be dedicated to access by end users initializing software for local markets, while a separate one or more database may be dedicated to developers building the internalized code.

[0045] Similarly, while the invention has in embodiments been described as generating translations and related data using one parsing engine 110, in embodiments the parsing logic and related hardware or software may be distributed among multiple servers or other resources. Other hardware, software or other resources described as singular may in embodiments be distributed, and similarly in embodiments resources described as distributed may be combined. Further, while illustrative text translations were generally described as mapping one textual string to one target textual string, in embodiments further translations to other languages may in instances be possible, or localization to multiple languages in one software package may be performed. The scope of the invention is accordingly intended to be limited only by the following claims.